**Smart contract lottery - Project setup**

**Getting Started**

Prepare yourself because this next project will be extremely awesome!

For the project, we'll be working with an advanced lottery or raffle smart contract. This won't just be an exercise in coding, but a chance to learn more about:

* Events
* On-chain randomness (done the proper way)
* Chainlink automation
* And many more!

Please follow Patrick's presentation on the project we are going to build. Marvel at how good the code looks, pay attention to code structure, Natspec comments and all the other cool features.

Hopefully, that sparked your interest. Now let's get cooking!

**Setup**

We are going to start a new foundry project. You already know how to do that, it would be great if you could do this on your own and then come back and compare your work to what's presented below. With that out of the way, please call the following commands in your terminal:

mkdir foundry-smart-contract-lottery-f23

cd foundry-smart-contract-lottery-f23

code .

Inside the new VSCode instance, in the terminal, we are going to init our Foundry project:

forge init

Please delete all the Counter files that are populated by default when we initiate a new Foundry project.

A good practice at this stage in your project is to come up with a plan/blueprint. What do we want to do with this? What is the main functionality of the project? In the root folder create a file called README.md (if Foundry created one for you just delete its contents), you obviously know what this file must look like from the previous courses, but before we get there, let's just outline a simple plan.

Open the README.md:

# Proveably Random Raffle Contracts

## About

This code is to create a proveably random smart contract lottery.

## What we want it to do?

1. Users should be able to enter the raffle by paying for a ticket. The ticket fees are going to be the prize the winner receives.

2. The lottery should automatically and programmatically draw a winner after a certain period.

3. Chainlink VRF should generate a provably random number.

4. Chainlink Automation should trigger the lottery draw regularly.

We will introduce the Chainlink integrations in future lessons. For now, remember that we will use the Chainlink VRF service to obtain a random number, an operation that is harder than you think in the absence of a Chainlink VRF-like service. We are going to use Chainlink Automation to schedule and trigger the lottery draw programmatically.

**Let the development begin!**

Inside the src folder create a file named Raffle.sol. Inside the newly created file, we start our new project as always:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.18;

contract Raffle {

}

As you might know already having a strong NATSPEC is a key element in developing a nicely structured and readable smart contract. Let's create a NATSPEC description above the contract declaration line:

/\*\*

\* @title A sample Raffle Contract

\* @author Patrick Collins (or even better, you own name)

\* @notice This contract is for creating a sample raffle

\* @dev It implements Chainlink VRFv2.5 and Chainlink Automation

\*/

contract Raffle {

}

Let's think about the structure of our project, what is the main functionality a raffle should have?

1. Users should be able to enter the raffle by paying a ticket price;
2. At some point, we should be able to pick a winner out of the registered users.

contract Raffle{

function enterRaffle() public {}

function pickWinner() public {}

}

Good! Given that users need to pay for a ticket, we need to define the price of this ticket and also make the enterRaffle function payable to be able to receive the user's ETH. Every time we introduce a new state variable we need to think about what type of variable we need to use. Should we make the entranceFee constant, immutable or simply private? Why private and not public? The best solution is to make it a private immutable, so we get to define it once at the constructor level. If we decide to create a new raffle we simply redeploy the contract and change the entranceFee. Ok, but we need people to be able to see what they should pay as entranceFee. To facilitate this we will create a getter function.

contract Raffle {

uint256 private immutable i\_entranceFee;

constructor(uint256 entranceFee) {

i\_entranceFee = entranceFee;

}

function enterRaffle() public payable {}

function pickWinner() public {}

/\*\* Getter Function \*/

function getEntranceFee() external view returns (uint256) {

return i\_entranceFee;

}

}

Did you spot that slick /\*\* Getter Function \*/. To make our contracts extremely tidy and greatly improve readability we should use a Contract Layout. But more about this in the next lesson!